Synchronous Elastic Circuits with Early Evaluation and Token Counterflow *

Jordi Cortadella
Universitat Politècnica de Catalunya
Barcelona, Spain

Mike Kishinevsky
Strategic CAD Lab, Intel Corp.
Hillsboro, OR, USA

Abstract

A protocol for latency-insensitive design with early evaluation and its implementation is presented. The implementation is based on a symmetric view of the system in which tokens carrying information move in the forward direction and anti-tokens canceling information move in the backward direction. When tokens and anti-tokens collide, they annihilate. The implementation is formally verified against the temporal properties of the elastic protocol and correct transfer of data. An example illustrates the flow for converting a regular synchronous design into the elastic form and demonstrates trade-offs in applying early evaluation and token counterflow.

1 Introduction

Synchronous elastic (or latency insensitive) systems have been suggested by a few research groups as a form of discretized asynchronous systems (see, e.g., [3,6,7]). Such systems are "elastic" in a sense that they can tolerate dynamic and static changes in latencies of computation and communication components. Therefore, they enable new micro-architectural trade-offs, for instance a wider use of variable latency components targeting average case optimization, rather than the worst case optimization traditional to regular synchronous circuits. They also enable correct-by-construction re-pipelining of wires and computation blocks – a useful feature that can simplify design and RC scaling in nano-scale technologies.

Conventional synchronous elastic systems rely on lazy evaluation: the computation is initiated only when all input data are available. Such behavior can be described with a Marked Graph, a subclass of Petri nets [10] without choices. Nodes represent functional units and data items are modeled as tokens on the arcs. When a node has tokens at *all* input arcs, it can consume all input data and produce the result on the output arcs.

The requirement that all inputs must be available to compute a result can be too strict. For example, if a functional unit computes $a = b * c$, it is not necessary to wait for both operands if one of them is already available and known to be zero. Therefore, the result $a = 0$ can be produced by an *early evaluation* of the expression. Consider a multiplexer, a typical digital circuit component, with the following behavior:

$$z = \texttt{if } s \texttt{ then } a \texttt{ else } b.$$

*This is an extended version of a paper published at the *Design Automation Conference*, San Diego, June 2007. This work has been partially supported by a grant from Intel Corp., CICYT TIN2004-07925 and a Distinction by the Generalitat de Catalunya.

Early evaluation can be applied if, for instance, s and a are available and the value of s is *true*. In that case, the result $z = a$ can be produced and the value of b can be discarded when it arrives at the multiplexer.

In early evaluation, care must be taken in preventing the spurious enabling of functional units when the *non-required* inputs arrive later than the completion of the computation. A possible technique is the use of *negative tokens*, also called *anti-tokens*. Each time an early evaluation occurs, an anti-token is generated at every non-required input in such a way that when it meets the positive token they annihilate. The anti-tokens can be *passive*, waiting for the positive token to arrive, or *active*, traveling in the backward direction to meet the positive token.

The idea of passive anti-tokens for early evaluation was used in [9,12], extending Petri nets for handling OR causality and nodes with arbitrary guard functions. [8] used a similar technique for performance estimation of elastic systems with early evaluation.

Early evaluation has also been used in asynchronous circuits. In [11], the inputs of blocks with early-evaluation are partitioned into early and late. Early evaluation is allowed when all early inputs have arrived. The block waits for all inputs to arrive before advancing to the next evaluation. Active anti-tokens have also been proposed by [1,2] for the design of faster asynchronous pipelines. In [1], special care was taken to avoid metastability regardless of the arrival order of signals.

This paper presents a behavioral model for synchronous elastic systems with early evaluation, called a *dual marked graph*, in which tokens and anti-tokens can travel in opposite directions. The protocol and circuit implementation of elastic systems with early evaluation based on this dual counterflow view is described. An implementation with a symmetry between the positive and the negative sub-systems is proposed. It is also shown that the passive anti-token is a particular case of the active one and can be derived by simplification. Special care was taken to formally verify the correctness of the implementation. The paper concludes with an example illustrating the elasticization flow, the performance analysis of early evaluation, and system performance vs. controller area trade-offs.

2 Model for early evaluation

This section presents a concurrent model for systems with early evaluation based on marked graphs. We assume the reader is familiar with the basic Petri net theory and refer to [10] for an excellent survey. The notation used in this paper is next presented.

A *marked graph* (MG) is a triple $G = (N, A, M_0)$, where N is a set of nodes, A is a set of arcs and $M_0 : A \to \mathbb{N}$ is a marking that assigns an initial number of tokens to each arc. Given a node n, the notation ${}^\bullet n$ and $n^\bullet$ is used to denote the set of incoming and outgoing arcs of n, respectively. Given a subset $\phi \subseteq A$, the total number of tokens of the arcs in ϕ at a given marking M is denoted by $M(\phi)$. A node n is *enabled* at a marking M if $M(a) > 0$ for every $a \in {}^\bullet n$. An enabled node n can fire producing a new marking M' such that

$$M'(a) = \begin{cases} M(a) - 1 & \text{if } p \in {}^\bullet n \setminus n^\bullet \\ M(a) + 1 & \text{if } p \in n^\bullet \setminus {}^\bullet n \\ M(a) & \text{otherwise} \end{cases} \quad (1)$$

Without loss of generality, we model elastic systems with strongly connected MGs (SCMG). For open systems interacting with an environment, it is possible to incorporate an abstraction of the environment into the model by a transition that connects the outputs with the inputs.

We next review a few properties of SCMGs [10]:

Token preservation. Let ϕ be a cycle of an SCMG. For every reachable marking M, $M(\phi) = M_0(\phi)$.

Liveness. An SCMG is *live* if every cycle, ϕ, is marked positively at M_0, i.e., $M(\phi) > 0$.

Repetitive behavior. A firing sequence σ from a marking M leads to the same marking *iff* every node from N fires the same number of times in σ.

2.1 Dual marked graphs

We extend the class of MGs by allowing negative markings and early enabling. We call this class *dual marked graphs* (DMG). In DMGs, a marking M is a mapping $M : A \to \mathbb{Z}$. A subset of nodes $E \in N$ is declared to be *early-enabling* (denoted with thicker bars). Given a marking M and a node n, the *enabling rules* for DMGs are defined as follows:

Positive (P) enabling: $M(a) > 0$ for every $a \in {}^\bullet n$. This is the conventional enabling condition.

Negative (N) enabling: $M(a) < 0$ for every $a \in n^\bullet$, i.e. all the successor arcs have negative tokens.

Early (E) enabling, for $n \in E$: $M({}^\bullet n) > 0$ and $M(a) = 0$ for some $a \in {}^\bullet n$, i.e. only some predecessor arcs have tokens.

While P- and N-enabling are defined for any node of a DMG, E-enabling is only defined for early-enabling nodes. E-enabling models computations that can start without having all the incoming data available. In a more detailed model, E-enabling is associated with an external guard that depends on data values, e.g. a select signal of a multiplexer or a zero flag for an input operand of a multiplier. In this paper we deal with a more abstract model that does not use external guards. Instead, we conservatively assume that an E-enabled node can fire non-deterministically when some of the inputs are available. This simplistic abstraction is sufficient for proving some important properties of DMGs that primarily depend on the firing rules of the system. The actual implementation of early enabling relying on these properties of DMGs will be discussed in Sect. 4.2.

A node is enabled in a DMG if it is P-, N- or E-enabled. Regardless of the enabling condition, the *firing rule* for DMGs is the same rule (1) used for MGs.

When an N-enabled node fires, it *propagates* the anti-tokens from the successor to the predecessor arcs. We call this phenomenon *token counterflow*. When an E-enabled node fires, it *generates* anti-tokens in the predecessor arcs that had no tokens.

Example. Fig. 1 depicts a DMG with one early-enabling node n_1 and three simple cycles: $C_1 = \{n_1, n_2, n_4, n_7\}$, $C_2 = \{n_1, n_3, n_5, n_7\}$ and $C_3 = \{n_1, n_3, n_6, n_8\}$. Every cycle has a token in the initial marking. Figure 1(b) depicts a reachable marking, with the symbol $\ominus$ representing anti-tokens. This marking can be reached from the initial one in Fig. 1(a) by firing nodes n_2 (P-enabling), n_1 (E-enabling) and n_7 (N-enabling). The firing preserves the sum of tokens at each cycle. For example, cycle C_1 has two positive tokens and one anti-token that sums up to one.

2.2 Algebraic properties of DMGs

Some of the fundamental properties of SCMGs also hold for strongly connected DMGs (SCDMG) since MGs and DMGs have the same firing rules. We will not give formal proofs for the properties, but an intuition sufficient for the purpose of this paper.

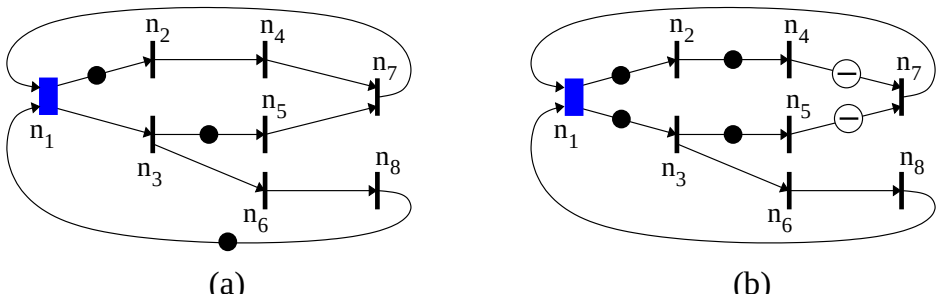

Figure 1: Dual marked graph: (a) initial marking, (b) reachable marking with anti-tokens.

Token preservation. Since the firing rule is identical, the token preservation property also holds in SCDMGs. For every cycle, the firing of a node does not change the *sum* of tokens that belongs to a cycle. Recall that in case of dual marked graphs some of the tokens can be positive and some negative.

Liveness. The reachable markings of an SCMG are also reachable in the SCDMG version. The token preservation property also guarantees that no deadlock can be produced even in the presence of negative tokens.

Repetitive behavior. From the firing rule it directly follows that a sequence in which all nodes are fired the same number of times leads to the same marking. Interestingly, the repetitive behavior is preserved regardless of the type of firing of the nodes (positive, negative or early).

The above properties are essential to guarantee a correct behavior in elastic systems with early evaluation.

3 Protocol for elastic communication

The protocol used for the synthesis of elastic circuits is based on the one presented in [6]. Two control signals, *Valid* (V) and *Stop* (S), determine three possible states of the channel (Fig. 2):

(T) Transfer, ($V \wedge \overline{S}$): the sender provides valid data and the receiver accepts it.

(I) Idle, ($\overline{V}$): the sender does not provide valid data.

(R) Retry, ($V \wedge S$): the sender provides valid data but the receiver does not accept it.

The sender has a *persistent* behavior when a *Retry* cycle is produced: it maintains the valid data until the receiver is able to read it. The language observed at a channel can be described by the regular expression $(I^*R^*T)^*$.

3.1 Implementation of elastic controllers

Fig. 3 depicts a latch-based elastic pipeline with the controllers for the data latches. Each controller (shadowed box) generates the enable signal (En) for one of the latches. This controller, together with the associated latch in the data-path, is called an *Elastic Half Buffer* (EHB). The labels associated to the latches, H (high) and L (low), indicate the active phase of the latch. For simplicity,

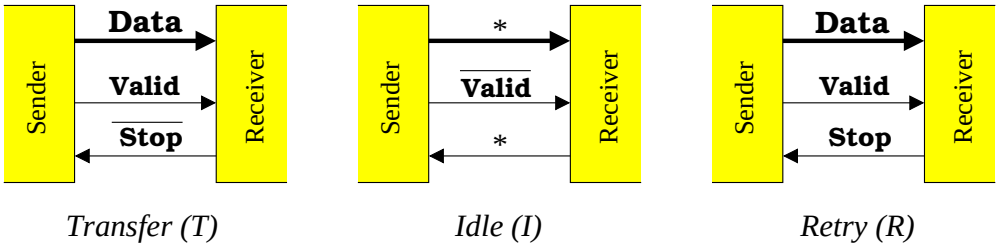

Figure 2: SELF protocol.

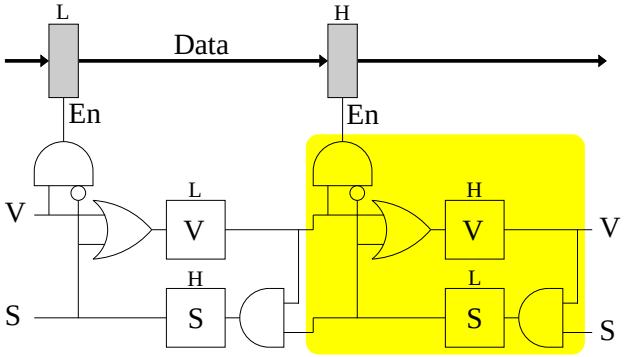

Figure 3: Linear pipeline with elastic half buffers.

the clock signal is omitted. The *Elastic Buffer* (EB) is a sequential composition of two EHBs (like a flip-flop is a composition of two transparent latches).

The protocol can be extended and implemented for arbitrary netlists by using the controllers for join and fork structures. The join controller (Fig. 4(a)) generates V at the output channel as the conjunction of the V's at the input channels. The S at the input is generated when no transfer is possible during the current cycle ($\overline{V} \vee S$).

The fork controller (Fig. 4(b)) has an eager behavior. The V_{out} signals are generated as soon as S_{out} is not asserted, regardless of the state of the other output channel. The gate R detects when a *Retry* cycle is produced. The flip-flops (FF) store the *retry* condition for the next cycle to keep signal V_{out} asserted through the gate V. The gate B detects when there are still some channels pending for a transfer. If so, no new V_{out} is generated for those channels that have already committed the transfer.

4 Elasticity with anti-tokens

The implementation of elasticity with anti-tokens is based on the formal model of DMGs presented in Sect. 2. In DMGs there are two flows, one for tokens and another for anti-tokens. A DMG can be split into two dual MGs, one for the flow of tokens and the other for the counterflow of anti-tokens. When a token and an anti-token are held in dual arcs, they must cancel each other.

The scheme of a linear pipeline is depicted in Fig. 5. The symbols V^+, S^+, V^- and S^- denote the *valid* and *stop* signals for the positive and negative flows. Note that the V^- signal has the semantics of a *Kill* for the positive tokens. The controllers are built based on the ordinary EHB

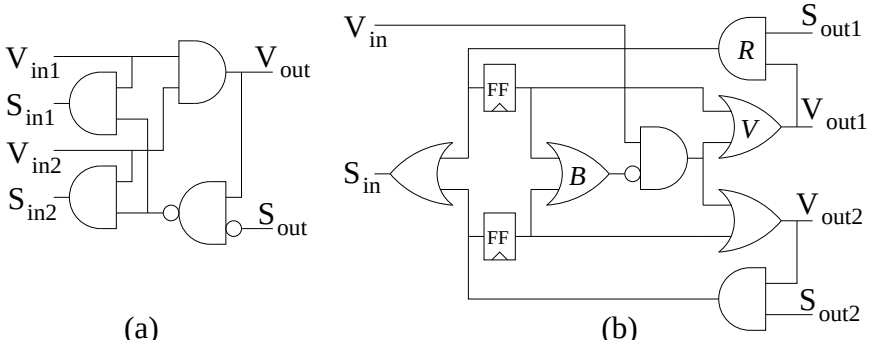

Figure 4: Elastic controllers: (a) Join and (b) Fork.

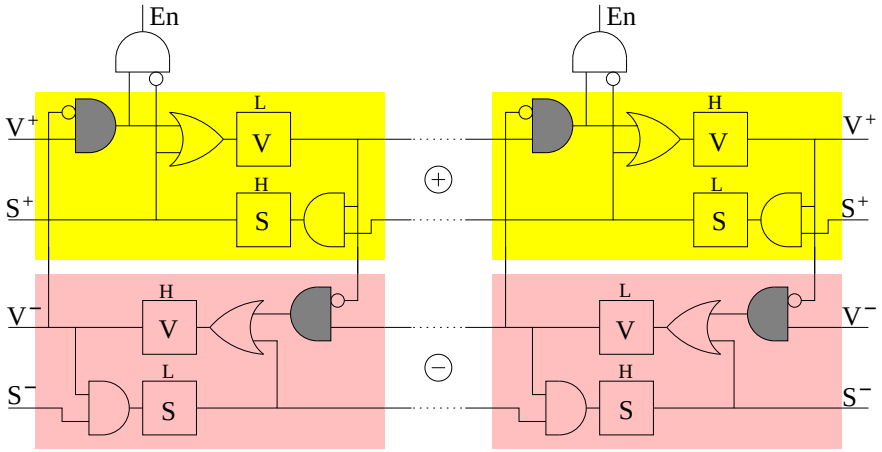

Figure 5: Linear pipeline with dual EHBs.

shown in Fig. 3. The negative component (bottom) is symmetric to the positive component (top). The dark gates are the only addendum to the controllers and they play the role of the mutual cancellation of tokens and anti-tokens when they meet on the same channel.

The abutment of elastic controllers with dual flow can easily lead to netlists with combinational cycles if controllers are not properly designed. For this reason, the gates cancelling tokens are placed at the boundaries of the EHB controllers, before the V and S signals are stored in their corresponding latches, as shown in Fig. 5.

The protocol for elastic communication implies certain invariants in the state of a channel. In particular:

$$\overline{V^- \wedge S^+} \qquad \text{and} \qquad \overline{V^+ \wedge S^-} \tag{2}$$

The first invariant indicates that it is not possible to kill a token and to stop it at the same time. The second invariant has a dual semantics for anti-tokens.

4.1 Join and fork controllers

The most difficult part of the implementation of elasticity is the correct synchronization of multiple flows by the *join* and *fork* structures (see Fig. 6(a) and 6(b)). The shadowed boxes of the join

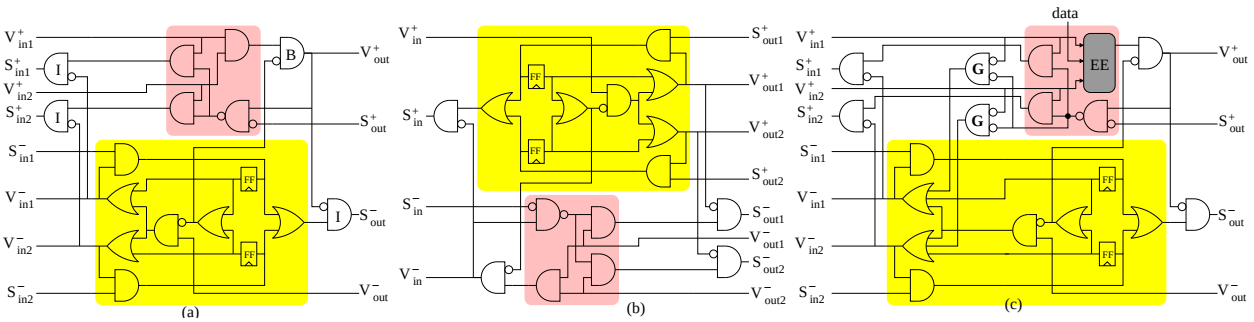

Figure 6: Dual elastic controllers: (a) Join, (b) Fork and (c) Join with early evaluation.

controller correspond to the basic join and fork controllers shown in Fig. 4. A join of tokens has a dual fork for anti-tokens, and vice-versa. The gates with label `I` are included to preserve the invariants specified by the expressions in (2). Finally, the gate `B` prevents a new transfer to occur before all the pending anti-tokens stored in the flip-flops FF have been propagated to the input channels.

Note the striking symmetry between the fork and the join controllers: after a half-turn Fig. 6(a) becomes identical to Fig. 6(b).

4.2 Join controller with early evaluation

The most important controller in this work is the one that generates anti-tokens. This corresponds to the *join with early evaluation*, depicted in Fig. 6(c). The design is similar to the join controller in Fig.6(a), with two important differences:

- The gates with label `G` generate the anti-tokens. For every input channel, they implement the equation

$$\overline{V_{in}^+} \wedge V_{out}^+ \wedge \overline{S_{out}^+}$$

  that feeds the OR gate producing the anti-tokens at V_{in}^-. This equation is asserted when there is a transfer at the output channel without data at the input channel.

- The shadowed block with label `EE` implements the early evaluation function using the V_{in}^+ signals and some data coming from the data-path. This function substitutes the conjunction of V_{in}^+ in the conventional join and may be asserted even though not all V_{in}^+ signals are true.

Example. A join controller for a 2-input multiplexer would have three input channels: s, a, and b. The first channel would be associated with the select signal, whereas the other two channels would be associated to the input data. Since every channel is elastic, it would also carry the V and S signals of the elastic protocol. The enabling function (block `EE`) could be:

$$\texttt{EE} = V_s^+ \wedge ((s \wedge V_a^+) \vee (\overline{s} \wedge V_b^+)).$$

The signal s corresponds to the data value of the channel with the same name. Note that V_s^+ must always be true for the enabling of the module. An early enabling is produced, for example when $V_s^+ = V_a^+ = 1$ and $s = 1$. In this case, if $V_b^+ = 0$, an anti-token will be produced in the channel b.

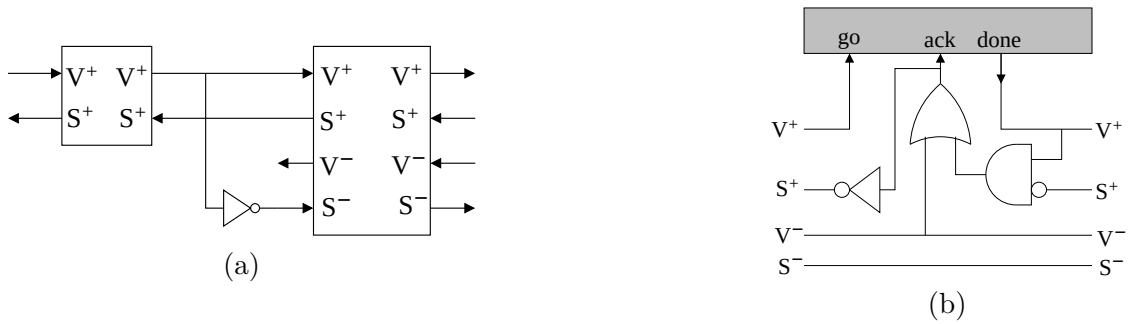

Figure 7: Control for (a) passive anti-tokens and (b) variable-delay units.

4.3 Constraint on enabling function

The following constraint on the early evaluation function should hold: each cofactor of `EE` with respect to *data* input must be *positive unate*. In other words `EE` makes decisions based on *presence* of valid inputs, not on their *absence*. In the considered example both cofactors are positive unate:

$$\texttt{EE}_s = V_s^+ \wedge V_a^+ \qquad \text{and} \qquad \texttt{EE}_{\overline{s}} = V_s^+ \wedge V_b^+.$$

4.4 Passive anti-tokens

Early evaluation is a feature that helps to improve the performance and reduce power in elastic systems. However the generation and propagation of anti-tokens is usually concentrated in very specific locations of the system. Handling the propagation of anti-tokens in the whole system can be a waste of area and energy.

Figure 7(a) depicts a mechanism to interface between those parts of the system that handle anti-tokens and those that do not. The strategy is simple: the propagation of anti-tokens can be stopped by asserting the S^- signal. In this way, when V^+ is not asserted, S^- stops the arrival of anti-tokens through V^-, thus creating the corresponding back-pressure for anti-tokens, if necessary. If V^+ is asserted, the tokens and anti-tokens are cancelled out. V^- is not used outside the controller. Note that the inverter maintains the channel invariant (2).

4.5 Variable latency

The controller for variable-latency units is presented in Fig. 7(b). There is a handshake protocol between the controller and the functional unit: the *Go* signal indicates a request to start the computation, the *Done* signal indicates the completion and the *Ack* signal indicates that the environment has accepted the result.

5 Formal verification

The design of elastic controllers is error-prone and requires formal verification. Since the size of the controllers is small, state-of-the-art model checking techniques readily apply.

Three different aspects of the implementation were verified: (1) the protocol is not violated, (2) the absence of deadlock, and (3) the correct interaction of the controllers with the data-path. CTL formulae [5] and `NuSMV` [4] were used to specify and verify temporal logic properties.

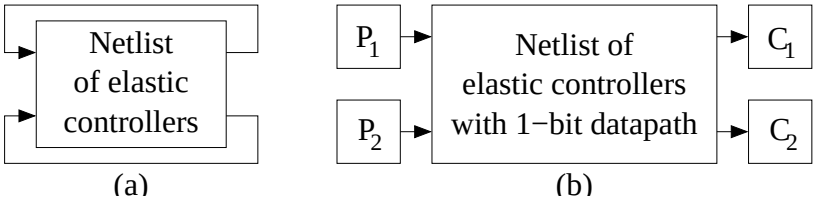

Figure 8: Formal verification set-up

The first two aspects were verified on several netlists explicitly designed to exercise different combinations of controllers. Care was taken to include feedbacks to verify that they do not introduce deadlocks (see Fig 8(a)). The controllers also included units with non-deterministic delays. The following CTL properties were checked for each channel:

$$\texttt{AG}\,((V^+ \wedge S^+) \implies \texttt{AX}\,V^+) \qquad \text{(Retry}^+\text{)}$$

$$\texttt{AG}\,((V^- \wedge S^-) \implies \texttt{AX}\,V^-) \qquad \text{(Retry}^-\text{)}$$

$$\texttt{AG}\,((\overline{V^+} \vee \overline{S^-}) \wedge (\overline{V^-} \vee \overline{S^+})) \qquad \text{(Invariant (2))}$$

$$\texttt{AG AF}\,((V^+ \wedge \overline{S^+}) \vee (V^- \wedge \overline{S^-})) \qquad \text{(Liveness)}$$

The first two formulae guarantee that the protocol at the positive and negative parts of the channel is persistent. The violation of one of these properties would mean that a trace different from $(I^*R^*T)^*$ could be produced (see Sect. 3). The third formula guarantees that the positive and negative part of the channel interact correctly by preserving the invariants specified by equation (2). Finally, the last formula guarantees liveness by checking that every channel will eventually see a token or anti-token in the future.

To check a correct interaction with the data-path, a class of systems like the one shown in Fig. 8(b) was constructed and verified. P_i and C_i denote producers and consumers of data, respectively. The netlist in-between is acyclic and initially contains no valid data. The data-path associated with the controllers is 1-bit wide and the producers generate an alternating trace of 0's and 1's. The consumers are designed to non-deterministically accept the incoming data or send anti-tokens to cancel the data in the netlist. In this way, all possible interactions of tokens and anti-tokens in the netlist are exercised. All the components are designed to have non-deterministic delays, so that any possible order of events is explored. A failure is produced when the consumer is not receiving an alternating trace of 0's and 1's. Every join present in the netlist is associated with a non-deterministic merge operation inside the 1-bit data-path. The merge is designed to produce a non-deterministic data when two incoming valid data with different values are received at the input channels. Join controllers with early evaluation are also included.

The above experiments were essential for a correct design of the controllers. The high variety of netlists and properties that were verified give a high confidence that the presented controllers, and their composition, are correct.

6 Example

We will use the example in Fig. 9(a) to illustrate early evaluation and the conversion of a synchronous system into elastic. The system has five functional units: S, I, F, M and W. Registers

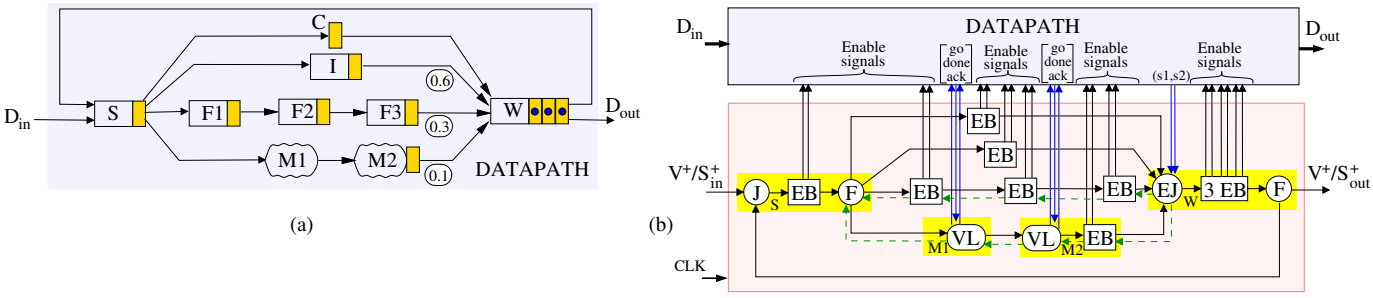

Figure 9: Example: (a) datapath, (b) elastic control.

are represented by shadowed boxes. Units F and W have three pipe-stages (the pipe-stages of W are not explicitly shown in the figure), while units S and I are not pipelined. Unit M is composed of two variable-latency multi-cycle units M_1 and M_2, delivering the result into a register. Unit S sends data to units I, F and M in parallel. Additionally, it sends control data (e.g. the opcode) to the register C. Unit W is a multiplexer that selects only one of the results according to the opcode. The selection probabilities are $0.6, 0.3$ and 0.1 for I, F and M, respectively.

The elastic conversion of this example (an automated process) proceeds in the following steps:

(1) All registers in the data-path are transformed into pairs of master-slave latches with independent enable signals.

(2) The elastic control layer (see Fig. 9(b)) that generates the enable signals for the latches of the datapath is built using the components presented in Sect. 3 and 4 as follows:

- For every register in the datapath, an EB controller (composition of two EHBs) is included in the control layer.
- For every block, a controller with a *join* (J) or *early join* (EJ) at the inputs and a *fork* (F) at the outputs is built. The join or fork components are omitted if the block has only one input or one output, respectively. It is the designer's responsibility to decide when to use early joins. In the example we chose to use an early join for module W to exploit the early evaluation of W's multiplexer. The operations are encoded with two control signals, s_1 and s_2, as follows: 00 for I, 01 for F and 1- for M, thus resulting in the following early-enabling function for W:

$$\mathtt{EE} = V_c^+ \wedge ((\overline{s_1} \wedge \overline{s_2} \wedge V_I^+) \vee (\overline{s_1} \wedge s_2 \wedge V_F^+) \vee (s_1 \wedge V_M^+)).$$

  The control signals s_1 and s_2 are bundled with the valid signal V_c^+ from register C.
- A variable-latency controller (VL) is included for each variable-latency unit (M_1 and M_2). Both of them have a three-wire (go, done, ack) interface with the data-path.
- The connection of the controllers with the $\{V^+, S^+, V^-, S^-\}$ interfaces is done according to the connectivity of the corresponding units in the datapath. Solid arcs represent pairs of $\{V^+, S^+\}$ wires in the positive sub-channels, while dotted arcs going in the opposite direction represent pairs of $\{V^-, S^-\}$ wires in the negative sub-channels. Some of the channels (e.g. $W \rightarrow S$), do not have a negative part, since both V^- and S^- signals are constant 0. This simplification is performed by simple logic synthesis techniques.

Configuration	Th	$Th(F_2 \to F_3)$ +	$Th(F_2 \to F_3)$ ±	$Th(F_3 \to W)$ +	$Th(F_3 \to W)$ −	$Th(S \to M_1)$ +	$Th(S \to M_1)$ ±	$Th(M_1 \to M_2)$ +	$Th(M_1 \to M_2)$ −	$Th(M_2 \to W)$ +	$Th(M_2 \to W)$ −	Area lit	Area lat	Area ff
Active anti-tokens	0.400	0.205	0.195	0.132	0.268	0.328	0.071	0.328	0.071	0.204	0.195	253	56	9
No buffer ($S \to W$)	0.343	0.116	0.227	0.106	0.237	0.285	0.058	0.285	0.058	0.228	0.115	241	52	9
Passive ($F_3 \to W$)	0.387	0.387	0.000	0.387	0.000	0.318	0.069	0.318	0.069	0.280	0.107	213	44	9
Passive ($M_2 \to W$)	0.280	0.143	0.137	0.095	0.185	0.280	0.000	0.280	0.000	0.280	0.000	234	52	9
No early evaluation	0.277	0.277	0.000	0.277	0.000	0.277	0.000	0.277	0.000	0.277	0.000	176	40	6

Table 1: Throughput for different configurations of the example in Fig. 9.

- In this example we assume that the environment has only $\{V^+, S^+\}$ interface and does not attempt to kill any tokens inside the system.

6.1 Synthesis and simulation

A complete framework for elastic systems has been designed. It can generate Verilog models for simulation, SMV models for verification and BLIF models for logic synthesis with SIS.

The Verilog model incorporates statements to randomly generate the values of the control signals according to the probability distributions defined by the user. Similarly, it also generates random delays for the variable-latency units. In the example, we define a latency for M_1 of 2 and 10 cycles with probabilities 0.8 and 0.2, respectively. The delay for M_2 is defined as 1 or 2 cycles with probability 0.5 each. Initially all three EBs at the output of W have valid data, represented by the tokens at the registers. The other EBs have bubbles (invalid data).

Table 1 summarizes results of 10K-cycle Verilog simulations and logic synthesis using SIS for five different configurations of the system. The second column shows the throughput of the system measured as the number of transfers per cycle at the interfaces with the environment.

The throughput of a channel is computed as the sum of *positive transfers* ($V^+ \wedge \overline{S^+} \wedge \overline{V^-}$), *negative transfers* ($V^- \wedge \overline{S^-} \wedge \overline{V^+}$) and *kill cycles* ($V^+ \wedge V^-$) and *is the same for all system channels*. This is a direct consequence of the *repetitive behavior* of SCDMGs (see Sect. 2.2). The next five columns show the throughput of positive transfers (+), negative transfers (−) and kills (±) for five selected channels of the system. The absent columns (e.g. ± in channel $F_3 \to W$) indicate that all values are 0. The last column shows the number of literals (factored form), transparent latches and flip-flops after logic synthesis.

The first configuration (active anti-tokens) corresponds to the one in Fig. 9(b): channel $F_3 \to W$ transfers anti-tokens 26.8% of the cycles, whereas channel $F_2 \to F_3$ kills tokens 19.5% of the cycles and has no anti-token transfers. The difference $26.8\% - 19.5\% = 7.3\%$ is the fraction of tokens that are killed on the internal channel between the two EHBs at the output of F_3.

The second line corresponds to a configuration that has no buffer on the channel $S \to W$. Interestingly, this degrades the throughput from 0.4 to 0.343 since long operations in the pipeline prevent S from producing new values for channel $S \to W$. This phenomenon occurs when there is a large mismatch among the latencies of different branches in a pipeline. The buffer C mitigates this phenomenon.

The third and fourth lines report results with passive anti-tokens (Fig. 7(a)) on one of the channels. This reduces the complexity of the control since some of the $\{V^-, S^-\}$ wires and their associated logic can be eliminated at the cost of some degradation in performance.

The numbers for $S \rightarrow M_1$ and $M_1 \rightarrow M_2$ are exactly the same, except for the fact that all negative tokens are transferred in the latter and killed in the former. This is because tokens are only killed at the boundaries with latches (see Fig. 5) and there are no latches between M_1 and M_2.

The last line reports a base-line performance for the lazy version of the control, in which the early join (EJ) is replaced by a regular join (J). In this case, no anti-tokens are held in the control layer.

This experiment illustrates the potential impact of early evaluation in throughput and control complexity. It also shows that the area overhead of the control layer is small for wide (e.g. 32 or 64-bits) datapaths.

7 Conclusions

This paper describes a particular implementation of early evaluation, but many variations are possible. For example, it would be possible to extend the approach to store multiple anti-tokens at every controller. This might improve performance in some corner cases, but we found little experimental motivation for this feature. The propagation of anti-tokens creates a distributed memory within the system and eliminates the need for storing them within early join nodes.

The mechanism for anti-token counter-flow can also be used for handling exceptions inside elastic pipelines. For example, flushing a pipeline on branch mispredictions can be done by injecting anti-tokens.

This paper has focused on the performance aspects of the early evaluation. The power reduction aspect due to disabling non-required activity in the data-path is an equally important side effect of the anti-token counterflow.

References

[1] M. Ampalam and M. Singh. Counterflow pipelining: Architectural support for preemption in asynchronous systems using anti-tokens. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 611–618, 2006.

[2] C. Brej. *Early Output Logic and Anti-Tokens.* PhD thesis, University of Manchester, 2005.

[3] L. Carloni, K. McMillan, and A. Sangiovanni-Vincentelli. Theory of latency-insensitive design. *IEEE Transactions on Computer-Aided Design*, 20(9):1059–1076, Sept. 2001.

[4] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In *14th International Conference on Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364. Springer-Verlag, 2002.

[5] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[6] J. Cortadella, M. Kishinevsky, and B. Grundmann. Synthesis of synchronous elastic architectures. In *Proc. ACM/IEEE Design Automation Conference*, pages 657–662, July 2006.

[7] A. Edman and C. Svensson. Timing closure through a globally synchronous, timing partitioned design methodology. In *DAC*, pages 71–74, 2004.

[8] J. Júlvez, J. Cortadella, and M. Kishinevsky. Performance analysis of concurrent systems with early evaluation. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, Nov. 2006.

[9] M. Kishinevsky, A. Kondratyev, A. Taubin, and V. Varshavsky. *Concurrent Hardware: The Theory and Practice of Self-Timed Design*. Series in Parallel Computing. John Wiley & Sons, 1994.

[10] T. Murata. Petri Nets: Properties, analysis and applications. *Proceedings of the IEEE*, pages 541–580, Apr. 1989.

[11] R. Reese, M. Thornton, C. Traver, and D. Hemmendinger. Early evaluation for performance enhancement in phased logic. *IEEE Transactions on Computer-Aided Design*, 24(4):532–550, Apr. 2005.

[12] A. Yakovlev, M. Kishinevsky, A. Kondratyev, L. Lavagno, and M. Pietkiewicz-Koutny. On the models for asynchronous circuit behaviour with OR causality. *Formal Methods in System Design*, 9(3):189–233, 1996.